DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MULTIPLE AUTHENTICATION

by
Onur ÇAKIRGÖZ

August, 2012
İZMİR

MULTIPLE AUTHENTICATION

A Thesis Submitted to the

Graduate School of Natural and Applied Sciences of Dokuz Eylül University

In Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering, Computer Engineering Program

by

Onur ÇAKIRGÖZ

August, 2012

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"MULTIPLE AUTHENTICATION"** completed by **ONUR ÇAKIRGÖZ** under supervision of **PROF. DR. SÜLEYMAN SEVİNÇ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Süleyman SEVİNÇ

Supervisor

Prof. Dr. Efendi NASİBOĞLU

(Jury Member)

Yrd. Doç. Dr. Ulaş BİRANT

(Jury Member)

Prof.Dr. Mustafa SABUNCU
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank to my thesis advisor Prof. Dr. Süleyman Sevinç for his help, suggestions and guidance.

I also thank to my family and my sincere friends for their patience and support.

Onur ÇAKIRGÖZ

MULTIPLE AUTHENTICATION

ABSTRACT

Authentication is one of the fundamental security mechanisms in computer science applications. Users can have access to the systems after authentication process is performed. Due to the easy use, passwords are mostly used for authentication. However, people encounter some problems with passwords in real life situations. One of the problem is users need to memorize and remember lots of passwords for distinct services. Unfortunately, rather than using disparate passwords, users generally prefer to use the same passwords for distinct services. Using the same password for different services give rise to security vulnerabilities. At this point, the question “Can we manage relatively strong and different passwords via a unique password?” arises.

In the scope of this study, an ancient theorem which is called Chinese Remainder Theorem was used to solve the problem. Firstly, a unique password was obtained from pre-defined passwords. But, since this unique password is very long and very difficult to memorize, another method has been developed. According to the second method, a unique password is defined by the user in advance then distinct and strong passwords are generated from the unique password. Finally, a secure multiple authentication protocol which is based on Chinese remainder theorem have been developed and the security analysis of the protocol have been done.

Keywords : Chinese remainder theorem, authentication, password, password reduction, password management

ÇOKLU KİMLİK DOĞRULAMA

ÖZ

Bilgisayar bilimi uygulamalarında kimlik doğrulama temel güvenlik mekanizmalarından bir tanesidir. Kimlik doğrulama işlemi gerçekleştirildikten sonra kullanıcılar sistemlere erişebilirler. Kimlik doğrulaması için kolay kullanımlarından ötürü çoğunlukla şifreler kullanılır. Fakat gerçek hayatta insanlar şifrelerle ilgili bazı problemlerle karşılaşıyorlar. Problemlerden bir tanesi kullanıcılar farklı servisler için birçok şifreyi ezberleme ve hatırlama ihtiyacı duymaktadırlar. Ne yazık ki farklı şifreleri kullanmak yerine kullanıcılar genellikle farklı servisler için aynı şifreyi kullanmayı tercih ediyorlar. Farklı servisler için aynı şifreyi kullanmak güvenlik zafiyetlerine neden olmaktadır. Bu noktada "Göreceli olarak güçlü ve farklı şifreleri tek bir şifre aracılığıyla yönetebilir miyiz?" sorusu ortaya çıkmaktadır.

Bu çalışmanın kapsamında, bahsi geçen problemi çözmek için Çinli Kalan Teoremi olarak adlandırılan eski bir teorem kullanılmıştır. İlk olarak, önceden tanımlanmış şifrelerden tek bir şifre elde edildi. Fakat bu tek şifrenin çok uzun olması ve ezberlenmesinin çok zor olmasından dolayı farklı bir yöntem geliştirildi. İkinci yönteme göre, tek bir şifre kullanıcı tarafından önceden belirleniyor daha sonra farklı ve güçlü şifreler bu tek şifreden üretiliyor. Son olarak, Çinli Kalan Teoremine dayanan güvenli çoklu kimlik doğrulama protokolü geliştirilmiş ve protokolün güvenlik analizi yapılmıştır.

Anahtar sözcükler : Çin kalan teoremi, kimlik doğrulama, şifre, şifre indirgeme, şifre yönetimi

CONTENTS

CHAPTER ONE

INTRODUCTION

Password, or formerly called parole, is an authentication method which is based on very old history. As is well known in general, a word agreed upon, or a character sequence is selected as a password and with the presentation of this password, the verification of the identity claim is performed. Some of the services that password authentication is used by are, e-mail servers, bank accounts, student accounts, numerous web sites, and so on.

Password has entered into our daily life with the widespread use of the internet. However the password usage that increases in daily life has provided deficiencies of this method to emerge noticeably. When users need to use more of the services requiring password, they are forced to memorize more and more passwords, as a result, they have begun to choose more simple and predictable passwords. Since the choice of simple passwords facilitates the work of malicious password hunters, institutions have defined constraints on the password's strength (predictability). The necessity which comes out with defining some constraints by service providers upon the strength(predictability) of passwords to be selected by the users, increases the requirement of more complex remembrance function. It is estimated that this contradictory situation causes some users to use very similar passwords even same password for different services. Thus, particular service provider's security policies that are applied to the user passwords and are aimed to be used only in its own service encounter the threat of losing the effects.

Authentication method via SMS that is commonly used today appears as a method which supports password-based authentication. Although this method does not increase the security of the password information theoretically, it emerges as an effective and deterrent method. Furthermore some methods such as a variety of one-time password applications, implementation of the obligation of replacing passwords periodically, using SSL(Secure Socket Layer) on the web, namely https, for storing passwords more secure - often running on mobile phone - password storage software have been widely taken in use to increase the security of password authentication.

Although authentication methods based on biometric characteristics of individuals proposed instead of authentication method via password, the password application did not lose its significance (Snelick, Uludag, Mink, Indovina & Jain, 2005), (Herley & Van Oorschot, 2012). Consequently, the techniques improving the usage security of password method are developed by researchers.

In this study, the method which is going to be devised removes difficulty in user's remembrance function and necessity of the usage of the similar passwords for distinct services. This is achieved with a method which is called password reduction. Simply, password reduction is defined as reducing n number of passwords defined for n number of service providers to a unique password through a mathematical procedure. Thus, without any loss of security, it is planned to increase the usability of password-based authentication systematic.

1.1 Recent Studies

1.1.1 Authentication using Smart Cards

Smart cards are widely used in remote authentication. Smart cards are preferred strongly by the users because of the reasons such as easy to use, mobility, efficiency, low computation cost and cryptographic preferences. Thus, many researches proposed smart card based authentication schemes such as (Yang & Shieh, 1999), (Hwang & Li, 2000), (Chien, Jan & Tseng 2002) and (Juang, 2004).

In smart card based authentication, firstly some information which corresponds to the user should be embedded into the smart card. This information is necessary for computations during the authentication session. This first phase is often called "registration phase". Registration phase is performed via an out-of-band(secure) channel.

After the first phase, smart card can be used by the users. To be able to provide the authentication, smart card should be placed into the card reader, and necessary information such as user-id and password should be submitted by the user. Card reader and smart card make pre-defined computations by using the information submitted by the user and the information embedded in the registration phase.

In this section, the scheme of (Shieh & Wang, 2006) will be explained as an example of the authentication methods using smart cards. (Shieh & Wang, 2006) have proposed an efficient remote mutual authentication and key agreement protocol using smart cards. The proposed protocol is computationally efficient and provides mutual authentication. It is efficient because the computations include one-way hash functions, XOR operations and concatenation operations. In the proposed scheme, time synchronization is not required although current time stamps are used as challenges and responses. Their protocol consists of two phases:

- The Registration Phase
- The Login and Key Agreement Phase

The symbols in their scheme are defined as in Table 1.1:

Table 1.1 The symbols used in Shieh & Wang's scheme

$h(\ )$	secure one-way hash function
x	the secret key maintained by the server
$\oplus$	exclusive-or operation
$\|$	string concatenation operation

Registration Phase

Assume a user U_i submits his identity ID_i and password PW_i to the server over a secure channel for registration. If the request is accepted, the server computes $R_i = h(ID_i \oplus x) \oplus PW_i$ and issues U_i a smart card containing R_i and $h(\ )$.

Login and Key Agreement Phase

When the user U_i wants to login to the server, he first inserts his smart card into a card reader then enters his identity ID_i and password PW_i. The smart card then carries out the following steps to begin an access session:

1. Calculate $a_i = R_i \oplus PW_i$.
2. Obtain current time stamp T_u, keep T_u in memory temporarily till the end of the session, and compute $MAC_u = h(T_u \parallel a_i)$.
3. Send the message (ID_i, T_u, MAC_u) to the server and wait for response from the server. If no response is received in time or the response is incorrect, send a failure report to the user and stop the session.

After receiving the message (ID_i, T_u, MAC_u) from U_i, the server performs the following steps to assure the integrity of the message, respond to U_i, and challenge U_i to avoid replay:

1. Check the freshness of T_u. If T_u has already appeared in a current executing session of user U_i, reject U_i's login request and stop the session. Otherwise, T_u is fresh.
2. Compute $a_i` = h(ID_i \oplus x)$ , $MAC_u` = h(T_u \parallel a_i`)$, and check whether $MAC_u`$ is equal to the received MAC_u. If it is not, reject U_i's login and stop the session.
3. Acquire the current time stamp T_s. Store temporarily paired time stamps (T_u, T_s) and ID_i for freshness checking until the end of the session. Compute $MAC_s = h(T_u \parallel Ts \parallel ai`)$ and session key $K_s = h((T_u \parallel T_s) \oplus a_i`)$. Then, send the message (T_u, T_s, MAC_s) back to U_i and wait for response from U_i. If no response is received in time or the response is incorrect, reject U_i's login and stop the session.

On receiving the message (T_u, T_s, MAC_s) from the server, the smart card performs the following steps to authenticate the server, achieve session key agreement, and respond to the server:

1. Check if the received T_u is equal to the stored T_u to assure the freshness of the received message. If it is not, report login failure to the user and stop the session.

2. Compute $MAC_s` = h(T_u \| T_s \| a_i)$ and check whether it is equal to the received MAC_s. If not, report login failure to the user and stop. Otherwise, conclude that the responding party is the real server.

3. Compute $MAC_u`` = h(T_s \| (a_i + 1))$ and session key $K_s = h((T_u \| T_s) \oplus a_i)$, then send the message ($T_s$, $MAC_u``$) back to the server. Note that, in the message (T_s, $MAC_u``$), T_s is a response to the server.

When the message (T_s, $MAC_u``$) from U_i is received, the server performs the following steps to authenticate U_i and achieve key agreement:

1. Check if the received T_s is equal to the stored T_s. If it fails, reject U_i's login request and stop the session.

2. Compute $MAC_u``` = h(T_s \| (a_i` + 1))$ and check whether it is equal to $MAC_u``$. If it is not, reject U_i's login request and stop the session. Otherwise, conclude that U_i is a legal user and permit the user U_i's login. At this moment, mutual authentication and session key agreement between U_i and the server are achieved. From now on, the user U_i and the server can use the session key K_s in their further secure communication until the end of the access session.

1.1.2 Secret Sharing and Asmuth-Bloom's Scheme

Secret sharing is a method which provides distribution of a secret amongst a group of participants. In secret sharing schemes, a dealer who is responsible for the distribution distributes shares of the secret to participants. The dealer gives only one share to each participant. Then, any group of t or more participants can reconstruct the secret. To reconstruct the secret, any t or more shares should be combined together. But no group of fewer than t participants can reconstruct the secret. This system is called a (t,n) threshold scheme. Secret sharing was invented independently by (Shamir, 1979) and (Blakley, 1979).

Secret sharing schemes are developed upon mathematical theorems. Thus, secret sharing can use Chinese Remainder Theorem. Because, from the definition of the Chinese Remainder Theorem, the unique solution can be thought as the secret and the simultaneous congruence equations can be thought as the shares. (Mignotte, 1983) and (Asmuth & Bloom, 1983) have developed (t,n) threshold schemes independently. Both of their schemes are based on the Chinese Remainder Theorem.

According to the (k,n) threshold scheme of (Asmuth & Bloom, 1983), firstly we choose integers k and n such that $n \geq 2$ and $2 \leq k \leq n$. Here, k denotes the minimum number of shares required to reconstruct the secret and n denotes the total number of shares. We generate a sequence of pairwise coprime integers such that $m_0 < \ldots < m_n$ and $m_0.m_{n-k+2}\ldots m_n < m_1\ldots m_k$. Then, the secret S can be chosen as a random integer in the set Z/m_0Z. After the selection of the secret S, we find a random integer α such that $S+\alpha.m_0 < m_1\ldots m_k$. To compute the shares $I_i = (s_i , m_i)$ we perform $(S+\alpha.m_0 \bmod m_i)$ for all $1\leq i \leq n$. If we want to reconstruct the secret S, firstly we combine any different k shares and solve the system of simultaneous congruences. Then the secret S can be computed as the unique solution of simultaneous congruences modulo m_0.

1.1.3 Password-Authenticated Key Exchange Protocols(PAKE)

Password-Authenticated key exchange protocols – sometimes called Password-only authenticated key exchange – require users to remember only a password. In these kind of protocols, public-private key pairs and symmetric(secret) key are not required to be stored. Bellovin and Merritt's password-based protocol (Bellovin & Merritt, 1992,1993) is the most well-known example of these. In their study, the problem of selecting poorly-chosen passwords has been addressed. Even if the situation where users select weak passwords, their protocol is secure against on-line and off-line dictionary attacks.

Although public/private key pairs and secret key are not needed to be stored, these keys have to be generated randomly by the system. The combination of asymmetric(public-key) and symmetric(secret-key) cryptography is used to provide secure communication over an insecure network. In their protocol, password is used to encrypt randomly-generated public key. The only information that the communicating parties have to share is password.

Their protocol is as following:

1. A sends A,P(E_A) to B.
2. B sends P(E_A(R)) to A.
3. A sends R(challenge$_A$) to B.
4. B sends R(challenge$_A$,challenge$_B$) to A.
5. A sends R(challenge$_B$) to B.

1.1.4 Federated Identity Management and SAML

Federated Identity management is the extension of classical identity management where enterprises or services exchange information between each other in accordance with pre-arrangements and pre-defined standards.

Identity management is a concept which provides centralized and automated management of identities. Rather than the classical approach where users are defined with identifiers(user-id), identity management approach presents identity and attributes associated with this identity as the main focus. According to the identity management concept, each user or process has to have a digital identity. Also this concept supplies a standard mechanism by which users verify their identities. By using identity management concept, users can have enterprise-wide access to resources in an authorized manner. The fundamental notion of an identity management system is the use of single sign-on(SSO). Single sign-on provides the advantage of enterprise-wide access of whole resources with a single authentication.

In identity management concept, users can create attributes which incorporate their digital identities. The responsible part of the identity management for the creation and maintenance of attributes is attribute service. Users can define their phone numbers, addresses, e-mail addresses as attributes. Attribute service enable users to define attributes once, so that this information is maintained in a particular place and released to data consumers when needed according to their authorizations.

Federated identity management provides multiple independent domains to exchange digital identities. The aim of the exchange of the digital identities between these distinct domains is to have an access to resources, services, applications across independent security domains by a user when a single authentication is performed. These domains include internal enterprise resources, external enterprise resources, other distinct services, applications. In order to exchange digital identities, cooperating enterprises should construct a federation based on the agreement and standards. Federated identity management includes standards, security policies and arrangements.

The underlying technology of federated identity management is SAML(Security Assertion Markup Language). SAML is an XML-based, open standard language which addresses the single sign-on problem on the internet. The OASIS Security Services Technical Committee started to develop a standard in January 2001 and published SAML v1.0 specification as an OASIS standard in November 2002. The latest version of Saml is v2.0 which was announced as an OASIS standard in march 2005.

In SAML identity provider(a producer of assertions) submits user's authentication request as an assertion to the service provider(a consumer of assertions) and in accordance with this assertion service provider makes a decision. As mentioned before, SAML is an XML-based technology and naturally SAML is constructed upon a number of existing standards such as XML Schema, XML signature and XML Encryption. Also SAML relies heavily on http as its communication protocol. Saml provides the exchange of the authentication and authorization information between online business partners in the form of assertions. Assertions consist of the three types of statements. These are:

- Authentication statements
- Attribute statements
- Authorization decision statements

1.1.5 Kerberos

Kerberos is a centralized authentication service which provides mutual authentication between user and server. Kerberos has been developed at MIT as a part of a project known as Athena (Miller, Neuman, Schiller, & Saltzer, 1987), (Steiner, Neuman, & Schiller, 1988), (Kohl, Neuman, & Tso, 1994). There are five versions of the Kerberos authentication service; version 1,2 and 3 are internal versions and are not used alone. Version 4 and version 5 take place in real-world distributed environments where security is a main issue. Kerberos and the protocol that it is based on are well-suited for an open distributed environment.

The secret key distribution scheme which has been developed by (Needham & Schroeder, 1978) is adopted as a base structure and Kerberos has been constructed upon this base. Their scheme involves the use of Key Distribution Center(KDC). The Key Distribution Center performs the responsibility of generating temporary keys(session keys) and distribution of these session keys. Each party has a master key which is shared with KDC. This master key is used to provide the security and confidentiality of session keys to be distributed.

The messages sent and received in a Kerberos authentication session are as following:

(1) C→AS $ID_c || ID_{tgs} || TS_1$

(2) AS→C $E(K_c,[K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$Ticket_{tgs}$ = E(K_{tgs}, [$K_{c,tgs}$||ID_c||AD_c||ID_{tgs}||TS_2||$Lifetime_2$])

(3) C→TGS ID_v||$Ticket_{tgs}$||$Authenticator_c$

(4) TGS→C E($K_{c,tgs}$, [$K_{c,v}$||ID_v||TS_4||$Ticket_v$])

$Ticket_{tgs}$ = E(K_{tgs}, [$K_{c,tgs}$||ID_C||AD_C||ID_{tgs}||TS_2||$Lifetime_2$])

$Ticket_v$ = E(K_v, [$K_{c,v}$||ID_C||AD_C||ID_v||TS_4||$Lifetime_4$])

$Authenticator_c$ = E($K_{c,tgs}$, [ID_C||AD_C||TS_3])

(5) C→V *Ticketv*||*Authenticatorc*

(6) V→C E($K_{c,v}$, [TS_5 + 1]) (for mutual authentication)

$Ticket_v$ = E(K_v, [$K_{c,v}$||ID_c||AD_c||ID_v||TS_4||$Lifetime_4$])

$Authenticator_c$ = E($K_{c,v}$,[ID_c||AD_c||TS_5])

The symbols used in the Kerberos protocol and their meanings can be seen from Table 1.2.

Table 1.2 The symbols used in the Kerberos protocol

Message (1) Client requests ticket-granting ticket	
ID_C	Tells AS identity of user from this client
ID_{tgs}	Tells AS that user requests access to TGS
TS_1	Allows AS to verify that client's clock is synchronized with that of AS
Message (2) AS returns ticket-granting ticket	
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2)
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key
ID_{tgs}	Confirms that this ticket is for the TGS
TS_2	Informs client of time this ticket was issued
$Lifetime_2$	Informs client of the lifetime of this ticket
$Ticket_{tgs}$	Ticket to be used by client to access TGS
Message (3) Client requests service-granting ticket	
ID_V	Tells TGS that user requests access to server V
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS
$Authenticator_c$	Generated by client to validate ticket
Message (4) TGS returns service-granting ticket	
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4)

Table 1.3 Continue

$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key
ID_v	Confirms that this ticket is for server V
TS_4	Informs client of time this ticket was issued
$Ticket_v$	Ticket to be used by client to access server V
$Ticket_{tgs}$	Reusable so that user does not have to reenter password
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket
ID_C	Indicates the rightful owner of this ticket
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_{tgs}	Assures server that it has decrypted ticket properly
TS_2	Informs TGS of time this ticket was issued
$Lifetime_2$	Prevents replay after ticket has expired
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tamperig
ID_c	Must match ID in ticket to authenticate ticket
AD_c	Must match address in ticket to authenticate ticket
TS_3	Informs TGS of time this authenticator was generated
Message (5) Client requests service	
$Ticket_v$	Assures server that this user has been authenticated by AS
$Authenticator_c$	Generated by client to validate ticket
Message (6) Optional authentication of server to client	
$K_{c,v}$	Assures C that this message is from V
$TS_5 + 1$	Assures C that this is not a replay of an old reply
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket
ID_C	Indicates the rightful owner of this ticket

Table 1.4 Continue

AD_c	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_v	Assures server that it has decrypted ticket properly
TS_4	Informs server of time this ticket was issued
$Lifetime_4$	Prevents replay after ticket has expired
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering
ID_C	Must match ID in ticket to authenticate ticket
AD_C	Must match address in ticket to authenticate ticket
TS_5	Informs server of time this authenticator was generated

1.1.6 Saravanakumar and Mohan's Single Password Protocol

(Saravanakumar & Mohan, 2008) have proposed a multiple authentication scheme which allows users to use the same user-id and the same password for distinct servers. Firstly, they have addressed the malicious server attacks, phishing attacks and the compromised server attacks. In malicious server attacks, an attacker can build up a malicious server which seems a legal server providing a particular service but actually it is intended to make use of gathering clients' passwords illegally. In most of the web sites, users have to reveal their passwords to authenticate themselves. Unfortunately an adversary who listens the communication between the user and the server can capture the user's password. This type of attack is called phishing attack. Saravanakumar and Mohan's multiple authentication scheme adopts the use of challenge/response and one-time server specific ticket to counter such types of attacks. In their scheme a user does not reveal his respective password at any point. Rather, the user uses his password with the challenge and the name of the server to generate the one-time server-specific ticket. The symbols in their scheme are defined as following:

Table 1.5 The symbols used in Saravanakumar & Mohan's protocol

C	Client or user-id
S	Server
P	Password
n_i , n_{i+1}	Challenges
MD()	Message Digest Function(One-way Hash Function)
MD_2()	MD(MD())
|	Concatenation

Their scheme consists of two phases. The scheme is as follows:

Registration Phase

Client generates a challenge n_i and ticket verification information $MD_2(n_i \mid p \mid s)$. Then client sends this information to the server for registration through a secure channel. Server stores this information to perform authentication process of the client later.

Login Phase

1. When client wants to login to the server, he sends his user-id C to the server.
2. Server sends the challenge which was generated by the client at registration phase.
3. Client creates one-time server-specific ticket $MD(n_i \mid p \mid s)$, new challenge n_{i+1} and new ticket verification information $MD_2(n_{i+1} \mid p \mid s)$ and sends these information to the server S.
4. Server S confirms the received ticket $MD(n_i \mid p \mid s)$ with the ticket verification information $MD_2(n_i \mid p \mid s)$. If the current ticket which Server S receives is valid, Server S authenticates the client C and immediately stores n_{i+1} in place of n_i and $MD_2(n_{i+1} \mid p \mid s)$ in place of $MD_2(n_i \mid p \mid s)$.

They adopt two assumptions for their protocol. Firstly, they assume that user remembers the password which consists of at least eight or more random characters. Secondly, they assume that their protocol is used with SSL(Secure Socket Layer).

1.1.7 Sevinç and Çakırgöz's Password Reduction Method

(Sevinç & Çakırgöz, 2012) have proposed 'Password Reduction Method' based on Chinese Remainder Theorem (CRT) and the Fundamental Theorem of Algebra (FTA). In this approach many passwords used for different services are reduced through a number theory procedure to a single password (call it X). The Password Reduction method can work in two directions; in the first case, called backward direction, a user has an existing set of n passwords (x_i) required to be reduced to a single one (X), in the second case, called forward direction, user starts with a single, easy-to-remember password (X) from which n passwords are generated each of which (x_i) is to be registered with a different service for authentication. In both cases, user needs only the single password (X) to authenticate with any of the n services. Password Reduction Method treats individual passwords as numbers ($\#x_i$ and #X represent number forms) equivalent to their string representation. In the backward direction, a random prime number (p_i where $p_i > \#x_i$) is generated for each of (x_i). It is intended to reduce n different passwords (x_i) to a singleone (X). Using ($\#x_i \bmod p_i$) n equations in CRT style are formed. It is a well-known fact that these n equations have a unique solution in modulo ($p_1p_2p_3\ldots p_n$) , call this product r. The unique solution to this equation system is the unique password (#X). Individual passwords (x_i) and their corresponding random prime numbers (p_i) are registered with each service. In addition to unique password, user also keeps a copy of the product of all primes (. This password and the product is used for securely logging in a service. At login time, user identifies herself to a service using a username then awaits the service to provide the prime number associated with her password. This prime number is used to ensure that the service is genuine as well as to generate the relevant password from previously computed single password. User generates password for the specific service using the single password computed earlier and the random prime provided at authentication time to the user by the service authenticating the user.

In the forward direction, user selects an easy to remember string (X) which then is used, in its number form, to generate n passwords (x_i) using CRT in the other direction. The end result in both cases is the same: unique password is the solution to a set of equations, each one representing one of n passwords to be reduced, as characterized by CRT.

The password reduction method, in its naive form, suffers from a weakness where an attacker can spoof a service and then provide a prime number (p) to a user with the intention of obtaining (#X mod p). By repeating this a few times, an attacker con construct a CRT like equation from which to predict the single password. To remedy this problem, authors in (Sevinç & Çakırgöz, 2012) have proposed that the product of all primes (r) used in Password Reduction Method to construct a CRT-like equation system be saved and used as benchmark by a user to check the authenticity of a service. Therefore, since each service is required to present a prime number (p_i) which they were given along with their individual passwords at the time of authentication, this prime number is obviously a factor of the product, i.e. p_i must divide r. The service authenticity can be verified by checking this fact. The authors refer to FTA for the security of this approach.

Our approach in this thesis is an enhancement of password reduction method of (Sevinç & Çakırgöz, 2012) and eliminates all known attack types as security threats to the method. We focus on our approach (Enhanced Password Reduction Method) in the following chapters.

CHAPTER TWO
ENHANCED PASSWORD REDUCTION METHOD

2.1 Passwords and Integers

The passwords used today are sequences consisting of symbols. These symbols can be letters, numbers and punctuation marks. On the one hand sequences to be selected as a password should be easy to remember, on the other hand the security of them should be strong in terms of service providers. Although passwords consisting of personal information such as name, surname or phone number are easy to remember for the user, they are classified as not secure passwords. Because they are also easy to estimate for password hunters. For example, password “sp961?&$icm” is difficult to remember for users but it emerges as a relatively high secure password. The strength of a password is related to its predictability. Assuming that passwords are selected totally randomly by users, it can be said that the strength of passwords is related to the number of symbols in the symbol space and the length of the password. For example, 10^{20} different passwords that contains 10 symbols can be constructed with the symbol alphabet which comprises 100 symbols. However, in practice, users' chosen passwords are not completely random, even if they are replaced (Gong, Lomas, Needham & Saltzer, 1993).

It is a well known fact that when s symbols exist in the symbol space, sequences consisting of these symbols can be expressed by a polynomial. Password $c_{n-1}c_{n-2}...c_1c_0$ can be expressed with the unique polynomial of $a_{n-1}s^{n-1} + a_{n-2}s^{n-2} +..+ a_1s^1 + a_0s^0$ numerically. Here, symbols are represented by c, the numeric values corresponding to these symbols in the Unicode Table are represented by a. For example a->97, b->98, c->99, ...etc. Each password can be converted into an integer with the calculation of this polynomial at point s. For instance, if it is assumed that there are total 100 symbols in symbol space, the password of abc is $1.100^2 + 2.100^1 + 3.100^0 =$ 10203. In other words the password of abc matches uniquely with the number of 10203. Furthermore when we have such an integer, the corresponding password of this integer can be obtained exactly and uniquely. Thus, it is possible to obtain the integer corresponding to a password or vice versa to obtain password corresponding to the integer.

As defined in the above expression calculating results of the polynomials, namely, for converting a password into an integer there is a method known as the Horner's rule method. This method significantly reduces the number of transactions made when calculating the result of a polynomial. However, since it is commonly known in the literature, it's details will not be described here. The conclusion reached here is, the password sequences can be addressed such as integers. This provides the use of all the mathematical methods applied to integers for the manipulation of passwords.

2.2 Formulation of the Problem

Let's suppose that a user determines a different password for each of the n electronic services. Here, our goal is to pass from the n different passwords to a unique password. Since the fact that each password corresponds to an integer, reduction of n integers that we have to a single integer can be expressed as the mathematical formulation of our problem. The mathematical formulation of the problem is expressed in equation(1). (Since user has determined n passwords, it is assumed that he knows the passwords and anyone other than himself knows the passwords.)

$$f: Z_n \rightarrow Z \quad \text{(Z: positive integers)} \qquad (1)$$

So, the problem of producing a single password from n passwords can be expressed as defining a function f between n-dimensional integer space and one-dimensional integer space as described above.

For example, function f can be defined as a simple arithmetic addition. In this case, value of the function f would be the sum of the all passwords. For instance if there are three passwords (n = 3), and if these passwords are 4, 7 and 8 the function f would generate 19. But when we have an integer 19 from here it is not possible to get 4,7 and 8. 12, 2, 5 and 14,3,2 will also result 19 when they are added. In this situation, it is clear that function f should be a reversible function. A reversible function can be defined as in equation (2).

$$f^{-1}: Z \to Z_n \text{ (Z: positive integers)} \quad (2)$$

such that,

$$f^{-1}(f (z_1,z_2,..,z_n)) = (z_1,z_2,..,z_n) \quad (3)$$

2.3 Chinese Remainder Theorem

The method that will be used in our thesis is based on an ancient theorem which is frequently used in number theory. This theorem is known as Chinese remainder theorem. This theorem has found place widely in the literature (Koblitz, 1994), (Ding, Pei, & Salomaa, 1996), (Cormen, Leiserson, Rivest, & Stein, 2001), (Iftene, 2007). Chinese Remainder Theorem was originated by a Chinese mathematician Sun Tzu. The first form of the Chinese Remainder Theorem was published in a third-century AD book(The Mathematical Classic by Sun Zi).

Chinese Remainder Theorem is about finding a solution to the system of simultaneous congruences. Suppose that X, a and p are positive integers. Then equation (4) defines a congruence.

$$X \equiv a \pmod p \quad (4)$$

A system of simultaneous congruences is defined in equation(5). Here $p_1, p_2, ..., p_n$ should be pairwise coprimes. Then, this system of simultaneous congruences has a unique solution X (mod r).

$$\begin{aligned} X &\equiv a_1 \pmod{p_1} \\ X &\equiv a_2 \pmod{p_2} \\ &... \\ X &\equiv a_n \pmod{p_n} \end{aligned} \tag{5}$$

Given,

$$r = \prod_{i=1}^{n} p_i \tag{6}$$

Let,

$$M_i = \prod_{j=1, j \neq i}^{n} p_j \quad (1 \leq i \leq n) \tag{7}$$

Then X is computed as in equation(8):

$$X = \left(\sum_{i=1}^{n} a_i \, M_i \, (M_i^{-1} \bmod p_i)\right) \pmod{r} \tag{8}$$

2.4 Backward Direction Method

Based on this theorem, we might think $(a_1, a_2, ..., a_n)$ as n passwords that we have. In response to these, prime numbers ($p_1, p_2 ... p_n$) that are greater than these numbers can be generated randomly by using known methods and can be used to acquire individual passwords. The solution of this sytem of simultaneous congruences would give us the X, namely the value of the unique password.

Extracting individual passwords from X is straightforward. In this case k'th individual password can be computed as $X \equiv a_k \pmod{p_k}$.

Then, what we need to obtain individual passwords from X are the value of X and the corresponding prime numbers. Obtaining k'th password by someone who has only X or only prime number p_k is not possible. When this informations are put together desired password can be easily acquired. But, having informations individually is not sufficient in order to obtain paswords. Then we can define required steps:

1. Convert n passwords into integers individually by using Horner method.
2. For each password, generate a prime number that is greater than password and distinct from each other.
3. Compute X from the equation system below:
   $X \equiv a_1 \pmod{p_1}$
   $X \equiv a_2 \pmod{p_2}$
   ...
   $X \equiv a_n \pmod{p_n}$
4. Store X and prime numbers separately. Remove a_i numbers.

2.5 Forward Direction Method

As mentioned previously, users define either similar passwords or same password for different service providers. The Backward Direction Method does not yield a solution to this problem. Because passwords here are defined by the users in advance and we know that users generally define similar passwords for different services. Also generated X is a very big integer and the string equivalent of X is not a memorable password.

However, when we think of the set of simultaneous congruences one more time, we can see that this can be also achieved. Firstly, instead of starting from passwords individually, user creates a X value which is sufficiently complex but memorable (We will use numerical equivalent of X but user can define this as a convenient string which consists of characters.). Secondly, sufficiently large n prime numbers are generated randomly. Individual passwords can be obtained as X mod p_k (for the k'th service). Then we can define the steps of the method:

1. Choose a strong password X.
2. Convert the string X into its numerical equivalent with Horner method.
3. Generate n random and distinct prime numbers.
4. Perform (X mod p_i) for $p_1, p_2..., p_n$.
5. Convert the results after modulo operation into their string equivalents. Use the results after conversion as passwords and then remove them.
6. Store X and prime numbers separately.

Here, the condition of selection of p_i's as prime numbers is a stronger condition than required. It is an adequate condition that p_i's should be pairwise coprime for the unique solution of the set of the equations which subjects to the explanations above. Namely, greatest common divisor; gcd(p_i, p_j) should be 1 for all $1 \leq i,j \leq n$ and $i \neq j$. Since the cost of the running time of the Euclid's GCD algorithm which takes place widely in the literature is limited to Θ (log n), there is no hesitation about the selection of p_i's correctly.

The simple authentication protocol with any service is as following:

Registration Phase

In the registration phase, user transmits his user-id ID, a_i and p_i to the server S_i for $1 \leq i \leq n$ over a secure channel. Server S_i stores ID, a_i and p_i in it's database for this user.

Login Phase

1. User U sends his user-id ID to the server S_i.
2. When server S_i receives the ID, it sends the corresponding prime number p_i to the user U.
3. After user U receives the prime number p_i from the server S_i, he performs X % p_i and obtains a_i. Then the user U transmits a_i to the server S_i.
4. Server S_i checks the received a_i with it's database. If they are equal, Server S_i authenticates the User U. Otherwise, it rejects the request and stops the session.

Unfortunately, despite the use of SSL or TLS, the simple authentication protocol depicted above is vulnerable to some attacks. These attacks are:

1. A malicious server S_i may send different coprimes to the user and may store the received a_i's. Then, it may try to compute X by using the (a_i , p_i) pairs.
2. Let k be a positive integer, then (X mod p_1) can be expressed as $X = kp_1 + a_1$. Thus, finding X and finding k are equivalent. Assume that a malicious server sends $2p_1$ to the user. If the remainder is still a_1, this shows us that k is an integer which is divisible by 2. Similarly, a malicious server can send $4p_1$. If the remainder is still a_1, this shows us that k is an integer which is divisible by 4. By this method, a malicious server can obtain information about the value of X such as it's prime factors.
3. If a malicious server sends a p_i which is bigger than the X, it can obtain X easily.

To cope with the vulnerabilities of the simple protocol, we have developed a secure and efficient protocol by using some cryptographic means such as one-way hash function, xor operation, asymmetric encryption, challenge/response. The symbols used in our scheme are defined in Table 2.1:

Table 2.1 The symbols used in our scheme

U	User
S_i	I'th server
ID	User-id
h( )	Secure One-way hash function(SHA-2)
$\oplus$	Exclusive-OR operation
X	Unique password
x	Half of the unique password
c_i,c_{i+1}	Challenges(Randomly generated integers between 7 and 10 digits)
N_1,N_2	Randomly generated Nonce values
a_i	I'th individual password for the i'th Server S_i
p_i	I'th individual prime number for the i'th Server S_i
PU_u	Randomly generated public key
PR_u	Randomly generated private key
SK	Symmetric key
PU_{si}	Public key for the Server S_i
PR_{si}	Private key of the Server S_i
D( )	Decryption
E( )	Encryption
%	Mod operator
||	Concatenation

Registration Phase

User U generates a c_i value and calculates $h(h(x \oplus a_i \oplus (c_i \| PU_{si})))$, $E(x, (p_i \oplus h(x \oplus ID)))$ and $h(x \oplus ID \oplus PU_{si})$. Here, $h(h(x \oplus a_i \oplus (c_i \| PU_{si})))$ is called verification information. $h(x \oplus ID \oplus PU_{si})$ is used instead of ID. Note that $h(x \oplus ID \oplus PU_{si})$ is specific to a particular server. It is assumed that this three calculated value and c_i are sent to the Server S_i over a secure channel for registration. The

Server S_i then stores this four information in it's database to authenticate the user U later on.

Login Phase

When user U wants to login to the Server S_i , he performs the following operations:

1. Generate public-private key pair(PU_u , PR_u) and N_1 randomly.
2. Compute $h(x \oplus ID \oplus PU_{si})$ which is used instead of ID.
3. Encrypt (N_1, $h(x \oplus ID \oplus PU_{si})$, PU_u) with the public key(PU_{si}) of the Server S_i.
4. Send $E(PU_{si}, (N_1, h(x \oplus ID \oplus PU_{si}), PU_u))$ to the Server S_i.

After the Server S_i receives the message, it performs the following operations:

1. Decrypt the received message. $D(PR_{si}, E(PU_{si}, (N_1, h(x \oplus ID \oplus PU_{si}), PU_u))) = (N_1, h(x \oplus ID \oplus PU_{si}), PU_u)$.
2. Generate a nonce value N_2 randomly.
3. Encrypt $(N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i)$ using the received PU_u.
4. Send $E(PU_u, (N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i))$ to the user U.

When the user U receives the message from the Server S_i, he performs the following steps:

1. Decrypt the received message with the private key PR_u which was generated before. $D(PR_u, E(PU_u, (N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i))) = (N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i)$.
2. Check N_1 for validity. If the received N_1 is not equal to the generated N_1, stop the session.

3. Otherwise, Decrypt $E(x, (p_i \oplus h(x \oplus ID)))$ with x. Since user U knows the x and ID, he can compute p_i with $(p_i \oplus h(x \oplus ID) \oplus h(x \oplus ID))$. Check the length and the primality of p_i. If p_i is not a coprime or the length of it is too long than it has to be, stop the session.
4. If p_i is valid, authenticate the server S_i, perform X % p_i and obtain a_i.
5. Calculate $h(x \oplus a_i \oplus (c_i \| PU_{si}))$.
6. Generate new challenge c_{i+1} and symmetric key SK randomly.
7. Compute the next verification information $h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si})))$.
8. Encrypt $(N_2, h(x \oplus a_i \oplus (c_i \| PU_{si})), c_{i+1}, h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si}))),SK)$ with PU_{si}.
9. Send $E(PU_{si}, (N_2, h(x \oplus a_i \oplus (c_i \| PU_{si})), c_{i+1}, h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si}))),SK))$ to the server S_i.

After the Server S_i receives the message from the user U, it performs the following steps:

1. Decrypt the message using the PR_{si}. $D(PR_{si}, E(PU_{si}, (N_2, h(x \oplus a_i \oplus (c_i \| PU_{si})), c_{i+1}, h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si}))),SK))) = (N_2, h(x \oplus a_i \oplus (c_i \| PU_{si})), c_{i+1}, h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si}))),SK)$.
2. Check N_2 for validity. If the received N_2 is not equal to the generated N_2, stop the session.
3. Otherwise, compute $h(h(x \oplus a_i \oplus (c_i \| PU_{si})))$ with the received $h(x \oplus a_i \oplus (c_i \| PU_{si}))$.
4. Check the computed value with the stored verification information. If they are equal, authenticate the user U. From now on, user and the server S_i can communicate by using SK. Otherwise, reject the authentication request.
5. If the user is authenticated, replace c_i with c_{i+1} and $h(h(x \oplus a_i \oplus (c_i \| PU_{si})))$ with $h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si})))$ immediately.

The computational cost of our protocol for the user-side and for the server-side is listed in Table 2.2.

Table 2.2 The computational cost of our protocol

	User		Server	
Registration Phase	**Encryption:**	1	**Encryption:**	-
	Decryption:	-	**Decryption:**	-
	Xor Operation:	6	**Xor Operation:**	-
	Hash Function:	4	**Hash Function:**	-
	Concatenation:	1	**Concatenation:**	-
	Random Number Generation:	1	**Random Number Generation:**	-
	Comparison:	-	**Comparison:**	-
	P_i Check:	-	**P_i Check:**	-
	% Operation	-	**% Operation:**	-
	Total:	13	**Total:**	-
Login Phase	**Encryption:**	2	**Encryption:**	1
	Decryption:	2	**Decryption:**	2
	Xor Operation:	7	**Xor Operation:**	-
	Hash Function:	5	**Hash Function:**	1
	Concatenation:	14	**Concatenation:**	6
	Random Number Generation:	5	**Random Number Generation:**	1
	Comparison:	1	**Comparison:**	2
	P_i Check:	1	**P_i Check:**	-
	% Operation:	1	**% Operation:**	-
	Total:	38	**Total:**	13

2.6 Security Analysis of Our Protocol

We assume that the messages of the protocol are submitted over a secure channel. Based on the assumption, we show that our authentication protocol does not cause any additional security risks when we analyze each of the attacks.

2.6.1 Message Replay Attack

In message replay attack, an adversary firstly listens to the communication between the user and the server and tries to capture the messages. Then, adversary attempts to login to the server by replaying the captured messages. Our authentication protocol is secure against message replay attack. Because in each session public-private key pair(PU_u,PR_u), symmetric key(SK) and N_1 are generated randomly by the user. Similarly, in each session N_2 is generated randomly by the server.

If the adversary replays the captured message $E(PU_{si}, (N_1, h(x \oplus ID \oplus PU_{si}), PU_u))$, he can not decrypt the coming message $E(PU_u, (N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i))$ which is encrypted by the server. Because the adversary does not know the private key(PR_u) which is required to decrypt the message. Furthermore, the adversary can not respond to the coming message $E(PU_u, (N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i))$ with the message $E(PU_{si}, (N_2, h(x \oplus a_i \oplus (c_i \| PU_{si})), c_{i+1}, h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si}))),SK))$ which was captured in the previous session. Because, (N_2, c_i) in the current session and (N_2, c_i) which was used in the previous session are different. Therefore, when the server decrypts the coming message $E(PU_{si}, (N_2, h(x \oplus a_i \oplus (c_i \| PU_{si})), c_{i+1}, h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si}))),SK))$ from the adversary, it can not validate N_2 and $h(x \oplus a_i \oplus (c_i \| PU_{si}))$. Naturally, the server S_i rejects the login request of the adversary.

2.6.2 Malicious Server Attack

In this type of attack, an adversary firstly sets up a server which seems legal. Next, he provides the registration of the users to the system by serving several services. But, the actual aims of the adversary are obtaining the passwords of the users and having access to bank accounts of the users or other important services by impersonating them.

Our authentication protocol is secure against malicious server attack. The first reason of being secure against malicious server attack is a user does not release his unique password and user-id to any server. Furthermore, he does not release x, a_i and p_i in an open format. Thereby, a server can not know X, ID, x, a_i and p_i. The second reason is a malicious server can not compute X, x, a_i and p_i from $h(x \oplus a_i \oplus (c_i \parallel PU_{si}))$, $h(h(x \oplus a_i \oplus (c_i \parallel PU_{si})))$, $E(x, (p_i \oplus h(x \oplus ID)))$ and $h(x \oplus ID \oplus PU_{si})$. The third reason is $h(h(x \oplus a_i \oplus (c_i \parallel PU_{si})))$, $E(x, (p_i \oplus h(x \oplus ID)))$ and $h(x \oplus ID \oplus PU_{si})$ are specific to a particular server. Thereby, a malicious server can not impersonate any of it's users to login to another server by using the users' authentication information in it's database.

2.6.3 Password Files Compromise Attack

The aim of this type of attack is obtaining the authentication information of the users such as password, user-id or ticket by stealing the password file of a server. Our protocol is secure against password file compromise attack. The reasons are similar to ones mentioned in the malicious server attacks. The first one is an adversary can not compute X, ID, x, a_i and p_i from $h(h(x \oplus a_i \oplus (c_i \parallel PU_{si})))$, $E(x, (p_i \oplus h(x \oplus ID)))$ and $h(x \oplus ID \oplus PU_{si})$. The second one is an adversary can not compute the required information $h(x \oplus a_i \oplus (c_i \parallel PU_{si}))$ that will be used for the next authentication process from $h(h(x \oplus a_i \oplus (c_i \parallel PU_{si})))$.

2.6.4 Message Log Compromise Attack

Some servers which carry out high security policies save sent and received messages in a message log file. In this type of attack, an attacker firstly steals the message log file. Then, he tries to acquire the passwords of the users or required information for authentication. An attacker can not decrypt the message $E(PU_{si}, (N_1, h(x \oplus ID \oplus PU_{si}), PU_u))$. Because he does not know the private key PR_{si} of the server S_i. Also, he can not decrypt the message $E(PU_u, (N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i))$. Because the required private key PR_u is known only by the user. Even if the attacker acquires the private key PR_{si}, attacker can not use $h(x \oplus a_i \oplus (c_i \| PU_{si}))$ to authenticate himself. The reason is that this information is used for only one time.

2.6.5 Offline Dictionary Attack

In Offline Dictionary Attack, an attacker listens to the communication between the user and the server and records the messages transmitted. Then, eavesdropping adversary tries to acquire the password of the user from observed transcripts of login sessions.

Our protocol is secure against offline dictionary attack. An attacker can not decrypt the message $E(PU_{si}, (N_1, h(x \oplus ID \oplus PU_{si}), PU_u))$. Because the private key PR_{si} is known only by the server S_i. Since an attacker does not know the private key PR_u, he can not decrypt the message $E(PU_u, (N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i))$ too. Even if he decrypts the messages, he can not gather any information about X, ID, x, p_i from $E(x, (p_i \oplus h(x \oplus ID)))$ and $h(x \oplus ID \oplus PU_{si})$. Similary, he can not decrypt the message $E(PU_{si}, (N_2, h(x \oplus a_i \oplus (c_i \| PU_{si})), c_{i+1}, h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si}))),SK))$. Even if he decrypts the message, he can not acquire any information about X, x, a_i from $h(x \oplus a_i \oplus (c_i \| PU_{si}))$. Furthermore, he can not derive the required information $h(x \oplus a_i \oplus (c_{i+1} \| PU_{si}))$ for the next authentication session from $h(h(x \oplus a_i \oplus (c_{i+1} \| PU_{si})))$.

2.6.6 Online Dictionary Attack

In this type of attack, an adversary pretends to be a legitimate user and attempts to login to the server repeatedly by trying each possible password from a dictionary.

Our protocol is secure against online dictionary attack. An attacker can not construct $h(x \oplus ID \oplus PU_{si})$ which represents the user-id. Because an attacker can not guess the value of x and the value of the ID at the same time. So, he can not pass to the next steps of the protocol. Also, an attacker has maximum three chances. After three unsuccessful attempts an attacker can not try more passwords.

2.6.7 Man-In-The-Middle Attack

In this type of attack, an attacker intercepts and modifies the messages sent between the user and the server. Then he acts as the user to the server or vice-versa by sending modified messages. The aim of an attacker may be obtaining unauthorized access or acquiring the password of the user.

The proposed protocol is secure against man-in-the-middle attack. An attacker can not decrypt the intercepted message $E(PU_{si}, (N_1, h(x \oplus ID \oplus PU_{si}), PU_u))$ which is sent to the server. Thus, he can not modify the intercepted message. Similarly he can not decrypt the intercepted message $E(PU_u, (N_1,N_2,E(x, (p_i \oplus h(x \oplus ID))),c_i))$ which is sent to the user.

2.6.8 Identity Protection

Our protocol provides identity protection. This is achieved by sending $h(x \oplus ID \oplus PU_{si})$ instead of only real identity(ID). Also, the pseudo identifications $h(x \oplus ID \oplus PU_{si})$ of the same user for different servers are different from each other. In each session $h(x \oplus ID \oplus PU_{si})$ is sent to the server S_i with two random values(N_1, PU_u) and in an encrypted form $E(PU_{si}, (N_1, h(x \oplus ID \oplus PU_{si}), PU_u))$. Thus, an attacker can not associate the different sessions belonging to the same user.

2.6.9 Mutual Authentication

In mutual authentication, both the user confirms the identity of the server and the server confirms the identity of the user. This is achieved with encryption, nonce values(N_1,N_2) and verification information $h(h(x \oplus a_i \oplus (c_i \parallel PU_{si})))$ in our approach. User confirms the identity of the server with N_1, and server confirms the identity of the user with N_2 and the verification information $h(h(x \oplus a_i \oplus (c_i \parallel PU_{si})))$. Since the user encrypts N_1 with the public key of the server S_i, only server S_i can decrypt and send back N_1 to the user. The value of $h(x \oplus a_i \oplus (c_i \parallel PU_{si}))$ can be computed only by the user. Because only user knows the value of x and a_i.

The following algorithms are used in our experimentations:

The Euclidean Algorithm: This algorithm is a recursive function which is used to determine the greatest common divisor of two integers. Suppose we have integers a and b, then the greatest common divisor of a and b is the biggest integer which divides both a and b. The greatest common divisor of two relatively prime integers is 1. In our application randomly generated integers are firstly tested with the miller-rabin algorithm. Secondly, they are tested in pairs with this algorithm to make certain the relatively primality of the integers in pairs. Because the miller-rabin algorithm is not a deterministic algorithm. If the returned value is 1 from this algorithm for all the integers in pairs, then this shows that these integers can be used in our method which is based on the Chinese Remainder Theorem.

The Miller-Rabin Algorithm: In our implementation it is necessary to select several very large prime numbers randomly. This algorithm is used to control a large number for primality. If the algorithm returns composite for an integer, then this integer is not prime with one hundred percent certainty. However, if the algorithm returns inconclusive, then this integer may be prime or not. Namely, there is no one hundred percent certainty about the integer’s being prime.

The Extended Euclidean Algorithm: The Extended Euclidean algorithm is an extension of the Euclidean algorithm. Let a and b be integers, this algorithm finds integers x and y such that x is the multiplicative inverse of a modulo b, and y is the multiplicative inverse of b modulo a. As seen from the equation 8, it is necessary for our method to obtain the multiplicative inverse of M_i modulo p_i. For this purpose the Extended Euclidean Algorithm is used in our implementation.

CHAPTER THREE
IMPLEMENTATION

In the scope of this study, three programs were developed. These are Backward Direction method, Forward Direction method and the server application. These programs were developed by using Visual Studio .NET technology and Visual C# programming language.

3.1 Forward Direction Method

The implementation of Forward Direction Method can be seen on Figure 3.1. The richtextbox that user can enter the unique password, the textbox that is used to specify the number of passwords which will be generated by the program and the Calculate button which triggers the system are situated under the Inputs groupbox. The richtextboxes which show integer equivalent of the unique password and the pairs of generated passwords and primes are situated under the Results groupbox.

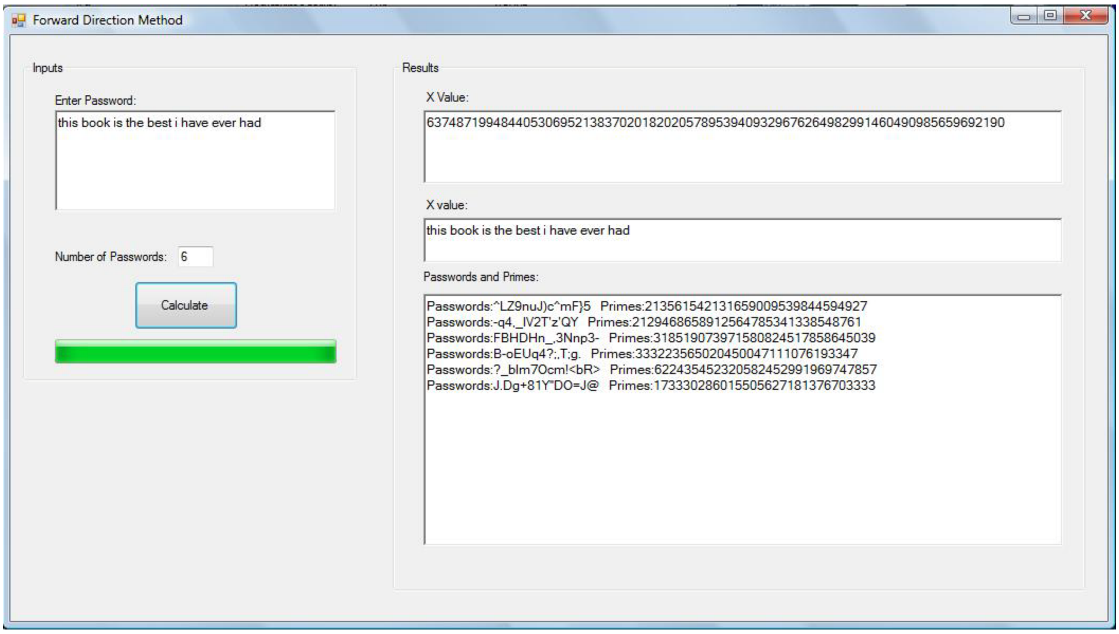

Figure 3.1 Forward direction method page.

If user clicks on the Calculate button after he enters the unique password and the number of passwords which will be generated, the program executes predefined methods and shows the generated passwords and primes on the screen. But if user makes a mistake, the program warns the user with an exclamation mark on the screen and do not execute the methods. The screenshot which includes warning can be seen on Figure 3.2. The mistakes are followings:

- Entering a unique password which includes Turkish characters
- Entering a unique password which is shorter than 30 characters
- Clicking on the button when there are empty fields

As seen from the Figure 3.1, the generated passwords consist of only keyboard characters for practical reasons. Because users generally do not prefer to use the characters which are not situated on keyboard. This situation is provided with a function which controls the generated passwords.

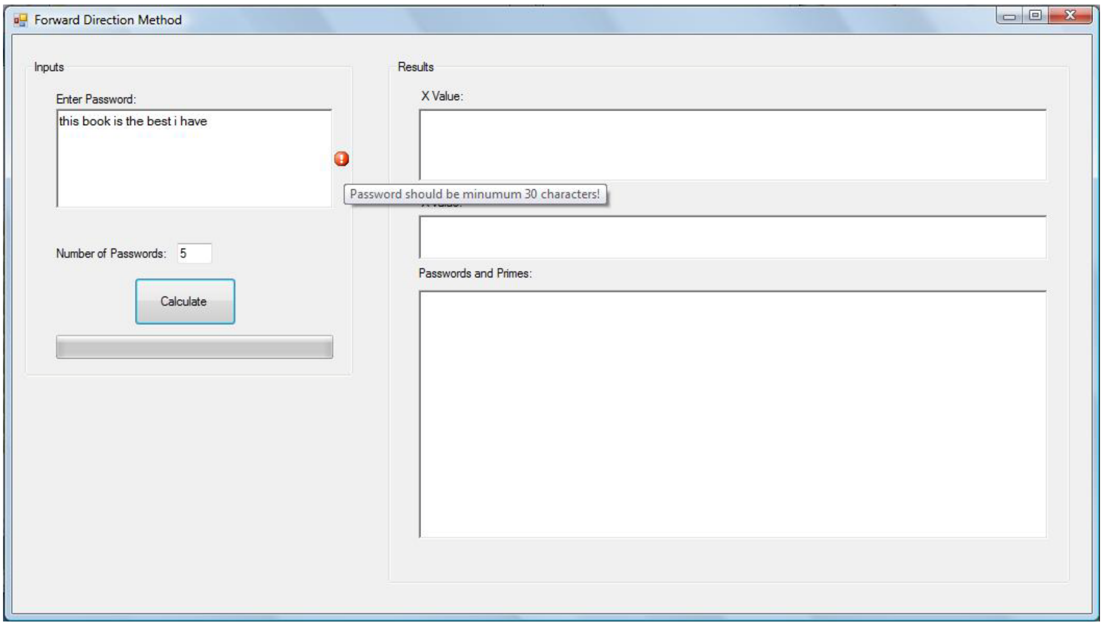

Figure 3.2 Forward direction method error page.

According to the ascii table on Figure 3.3, if the corresponding decimal values of all the characters which constitute a password is in [33..125], this string is accepted as a password. Also a password should contain minimum one lower character, one upper character, one digit and one punctuation mark.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figure 3.3 Ascii table.

3.2 Backward Direction Method

The implementation of Backward Direction Method can be seen on Figure 3.4. In the Generate X tab of the Backward Direction Method implementation, user firstly specifies the number of servers. After entering the number of server, equal number of textboxes for the names of the servers and the equal number of textboxes for passwords become visible on the page. Then user fills the textboxes. Finally, when user clicks on the Generate button, program executes pre-defined methods, writes the names of the server to a file and prints out X value(integer), X value(string), and primes in the richtextboxes. Furthermore numeric values of individual passwords, Mi values, inverse of Mi values and M value are printed out in the richtextbox which is situated at the bottom of the Results groupbox. Note that the generated unique password namely X value is not a memorable password. Another issue is that although there are three passwords, the system produces equal number of random passwords and primes and calculates X value according to the total six passwords and primes.

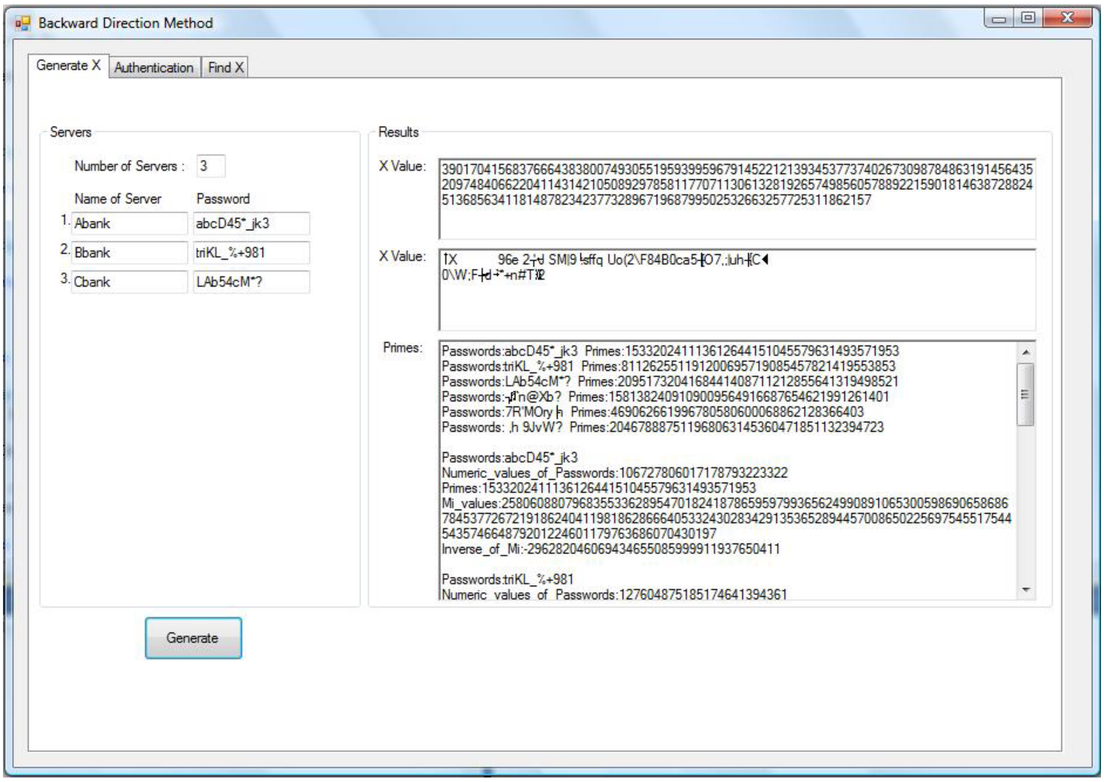

Figure 3.4 Backward direction method page(Generate X).

When user clicks on the Generate button without filling the required fields, the program shows an error message and warns the user. An example of this situation can be seen on Figure 3.5.

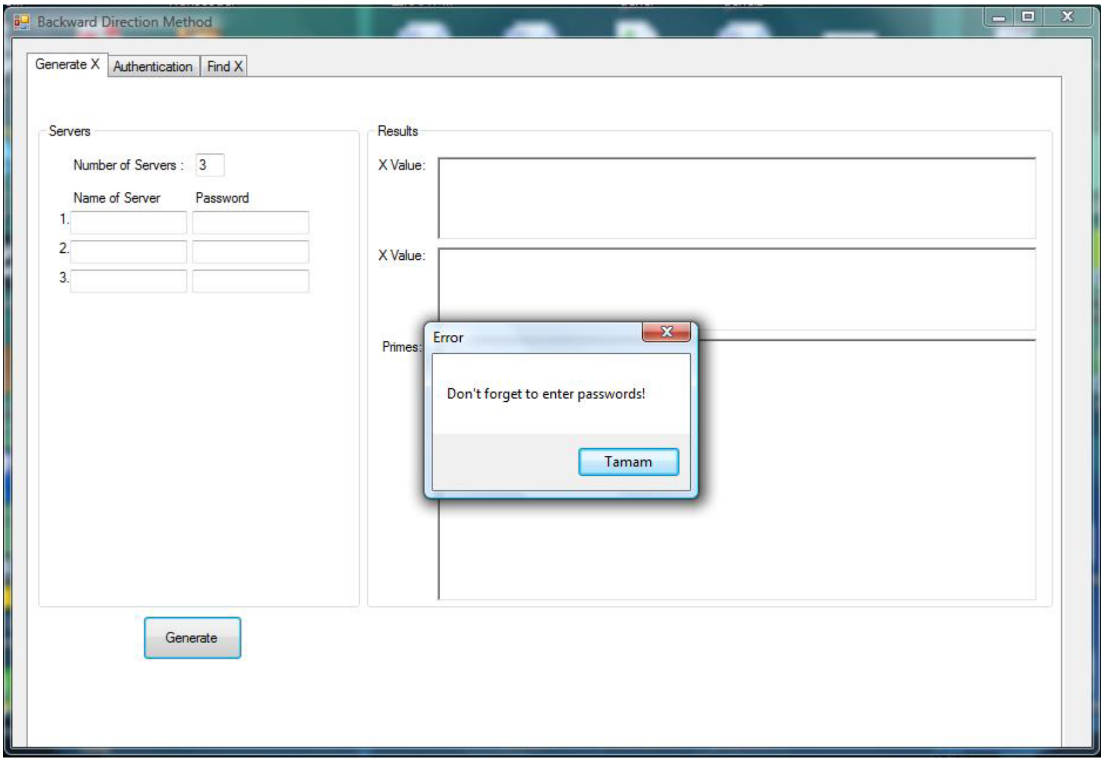

Figure 3.5 Backward direction method error page.

On Figure 3.6 the authentication tab of the Backward Direction Method is seen. At the first click of the Authentication tab, the program reads the names of the servers from the file, adds them to the CheckedListBox and shows it on the page. This Authentication tab is used to simulate the authentication process. Since we perform this simulation on the same computer, user can authenticate himself to only one server.

When user intends to authenticate himself to a server, firstly he should enter his user-id and unique password, then he should tick off the name of the server which he wants to authenticate and finally he should click on the Authenticate button. After user clicks on the Authenticate button, the program starts interaction with the server application and performs sending and receiving messages in accordance with authentication protocol. The authentication protocol is as follows:

1. User sends his user-id to the server.
2. Server sends the prime number which corresponds to received user-id to the user.

3. User calculates (X % prime number) and send the result(password) to the server.
4. Server checks the password. If password is valid server authenticates the user. Otherwise, server rejects the request.

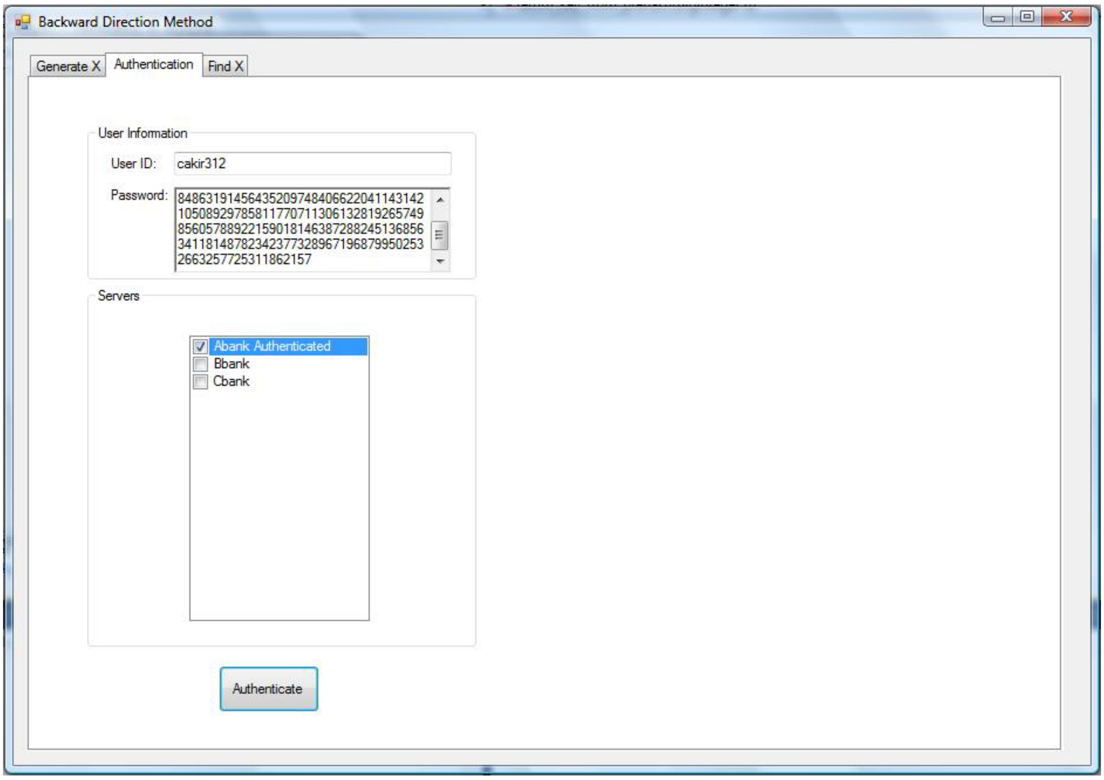

Figure 3.6 Backward direction method page(Authentication).

The last tab of the Backward Direction Method is Find X tab. This page is seen on figure 3.7. In the Find X tab of the Backward Direction Method, user firstly specifies the number of servers. After entering the number of server, equal number of textboxes for the passwords and the equal number of textboxes for prime numbers become visible on the page. Then user fills the textboxes. Finally, when user clicks on the Find X button, program executes pre-defined methods, finds the unique password, and prints out X value(integer) and X value(string) in the richtextboxes.

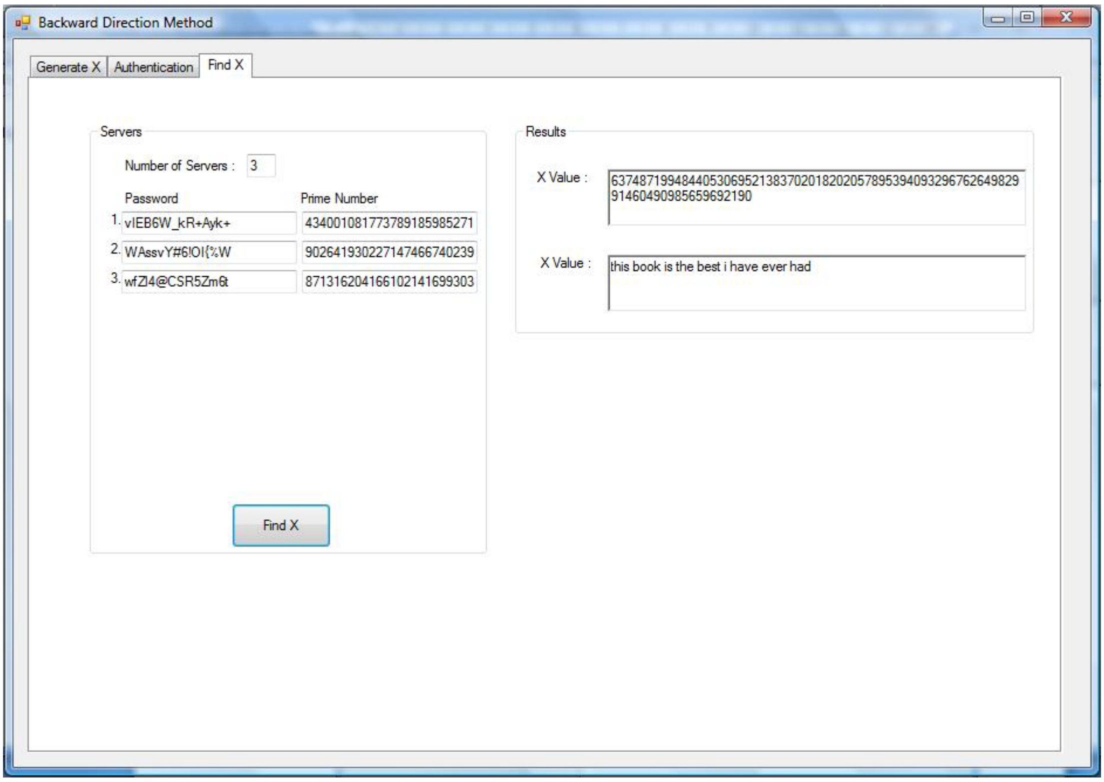

Figure 3.7 Backward direction method page(Find X).

3.3 Server Application

The server application can be seen on Figure 3.8. Server application is used to perform the process of the simulation of authentication. As mentioned before, server application interacts with the Backward Direction Method. When we execute server application, it starts listening to the coming requests. If an authentication request comes, program finds the prime number which corresponds to the received user-id from a text file and sends it to the user. After sending the prime number, if user sends the correct password server authenticates the user. To test the program locally, the ip address is set to “127.0.0.1” and port no is set to 20000. If any authentication request is accepted, the program specifies the acceptance with a message.

To stop the listening and to close the application, user should click on the Close button.

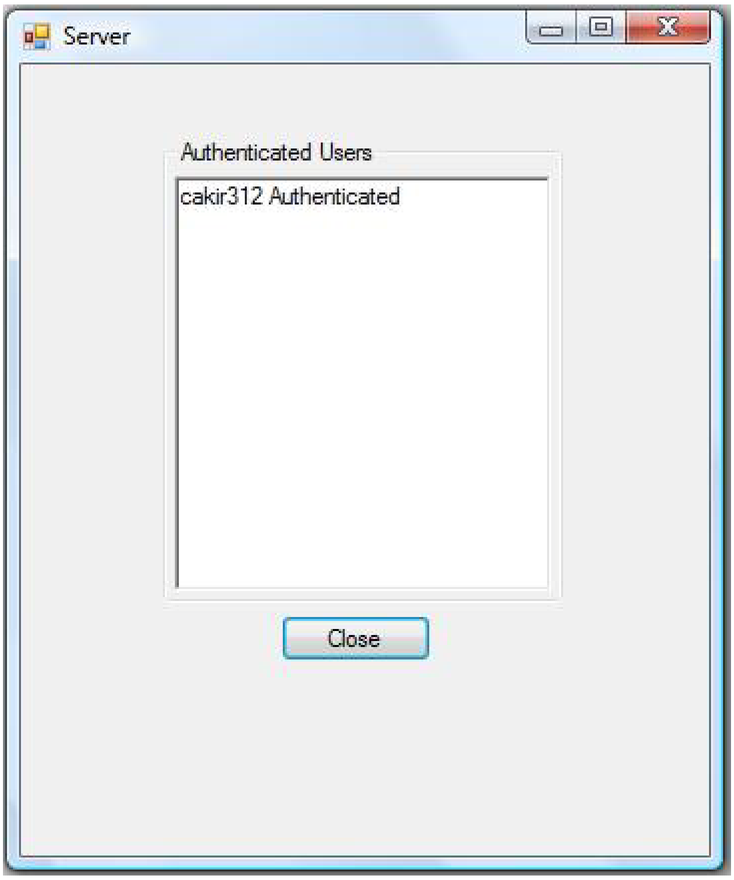

Figure 3.8 Server application page.

CHAPTER FOUR
CONCLUSION & FUTURE WORK

In the scope of this thesis, firstly the problem of managing and securing the lots of passwords created for different servers has been addressed. To solve this problem, Chinese Remainder Theorem is used in two different ways. In Backward Direction Method, we have seen that the generated password X, namely the unique password which a user should remember is not a memorable password. Then, we have developed the Forward Direction Method.

In Forward Direction Method, firstly the unique password is defined by the user. Then individual passwords are computed according to the X and the randomly generated prime numbers. The generated individual passwords are minimum 13 characters and consist of letters, digits, punctuation marks and mathematical operators. The experimental results where the length of the symbol space is set to 127, the length of the unique password is set to 30 and the length of the individual passwords is set to 13 can be seen on table 4.1. The important part of the table 4.1 is the difference between the minimum password and the maximum password. This difference shows us that 1632546855139074680584596572 different passwords which consist of 13 characters can be generated.

Based on the Forward Direction Method, the simple authentication protocol is created firstly. But, this simple protocol includes some security vulnerabilities. An attacker can gather information about the value of X from the vulnerabilities of the simple authentication protocol. Thus, we realized that this simple authentication protocol can not be used.

We have developed a secure and efficient authentication protocol which eliminates the security vulnerabilities of the simple protocol and which is resistant to all of the known attacks. According to our authentication protocol, a user can communicate securely with a server over a secure band. The other advantages of our protocol are a user never reveals his unique password(X), his user-id(ID), individual passwords(a_i) and prime numbers(p_i). So, we have achieved managing many passwords via a unique password. Also we have achieved authentication with multiple servers via same password and same user-id in a secure manner.

Our multiple authentication protocol is efficient. Because if we count only the number of high cost operations(encryption, decryption, hashing), server-side performs 4 operations in the login phase of the authentication protocol. Similarly, user-side performs 9 high cost operations in the login phase. The number of operations in the registration phase is not important. Because this phase is performed only one time.

Our future research intends to test our Enhanced Password Reduction multiple authentication method. This requires developing a stand-alone authentication module based on Enhanced Password Reduction method and a set of live services which can interact with the authentication module as described in this thesis. Although we developed a simulation system to see the effectiveness of our method, we hope to be able to test the method more thoroughly in a real environment.

Table 4.1 Experimental results

The Length of Symbol Space(s)	127		
The Length of Unique Password	30		
The Length of Individual Passwords	13		
Minimum Password	“!!!!!!!!!!!!!”	**Integer Equivalent of Minimum Password**	5855874589085811354270835 53
Maximum Password	“}}}}}}}}}}}}}”	**Integer Equivalent of Maximum Password**	2218134314047655816011680125
The Difference between Maximum and Minimum Passwords	1632546855139074680584596572		
Minimum Unique Password	“ “	**Integer Equivalent of Minimum Unique Password**	3302866817687290732042423259460853917204686118398286113758863 36
Maximum Unique Password	“}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}”	**Integer Equivalent of Maximum Unique Password**	1290182350659097942204071585726896061408080514999330513187056000
The Difference between Maximum and Minimum Unique Passwords	959895668890368868999829259780810669687611903159501901811169664		
Minimum Unique Password / Maximum Password	14890292245920008923676260942557416 6		
Maximum Unique Password / Minimum Password	220322742748579677492842402677518593 9		

REFERENCES

Ascii Table. (n.d.). Retrieved March 23, 2012, from http://en.wikipedia.org/wiki/File:ASCII-Table-wide.svg

Asmuth, C. A., & Bloom, J. (1983). A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29*(2)*, 208-210.

Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. (n.d.). Retrieved March 15, 2005, from http://docs.oasis-open.org/security/saml/v2.0/

Bellare, M., & Rogaway, P. (1994). Entity Authentication and Key Distribution. *CRYPTO '93 Springer-Verlag, Berlin, LNCS 773*, 232-249.

Bellovin, S., & Merritt, M. (1992). Encrypted Key Exchange: Password- Based Protocols Secure Against Dictionary Attacks. *In: Proc. IEEE Symposium on Research in Security and Privacy,* 72-84.

Bellovin, S., & Merritt, M. (1993). Augmented Encrypted Key Exchange: A Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise. *In: Proc. ACM. Computer and Communication Security,* 244-250.

Blakley, G. R. (1979). Safeguarding cryptographic keys. *Proceedings of the National Computer Conference. AFIPS Conf. Proc.*, 48, 313-317.

Chien, H. Y., Jan, J. K., & Tseng, Y. H. (2002). An efficient and practical solution to remote authentication: smart card. *Computers and Security*, *21*(4), 372–375.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). Introduction to Algorithms, *Second Edition. MIT Press and McGraw-Hill*, *Section 31.5: The Chinese remainder theorem,* 873–876.

Ding, C., Pei, D., & Salomaa, A. (1996). Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography. *World Scientific Publishing*, 1-224.

Gong, L., Lomas, M., Needham, R., & Saltzer, J. (1993). Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications, 11*(5), 648-656.

Herley, C., & Van Oorschot, P.C. (2012). A Research Agenda Acknowledging the Persistence of Passwords. *IEEE Security and Privacy Magazine, 10*(1), 28-36.

Hungerford, T. W. (1974). Algebra. *Springer-Verlag*, 131-132.

Hwang, M. S., & Li, L. H. (2000). A new remote user authentication scheme using smart cards. *IEEE Transactions on Consumer Electronics, 46*(1), 28-30.

Iftene, S. (2007). General Secret Sharing Based on the Chinese Remainder Theorem with Applications in E-Voting. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 186, 67–84.

Juang, W. S. (2004). Efficient password authenticated key agreement using smart cards. *Computers and Security, 23*(2), 167-173.

Knuth, D. (1997). The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Third Edition. Addison-Wesley, *Section 4.3.2*, 286-291.

Koblitz, N. (1994). A Course in Number Theory and Cryptography, *Springer*, 21.

Kohl, J. T., Neuman, B. C., & Tso, T. Y. (1994). The evolution of the Kerberos authentication system, Distributed Open Systems. *IEEE Computer Society Press*, 78-94.

Mignotte, M. (1983). How to share a secret. *Lecture Notes in Computer Science, Springer-Verlag*, 149, 371-375.

Miller, S. P., Neuman, B. C., Schiller, J. I., & Saltzer, J. H. (1987). Section E.2.1: Kerberos Authentication and Authorization System. *M.I.T. Project Athena, Cambridge, Massachusetts.*

Needham, R. M., & Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Comm. ACM*, 2, 993-999.

Rosser, B. (1941). Explicit bounds for some functions of prime numbers. *Amer. J. Math*, *63*, 211-232.

Saravanakumar, E., & Mohan, A. (2008). Single password multiple accounts. *Proceedings of the 2008 International Conference on Computing, Communication and Networking (ICCCN),* 1-7.

Sevinç, S., & Çakırgöz, O. (2012). Enhancing Password Security and Usability via Password Reduction Method. *(obtained through personal communication).*

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22, 612-613.

Shieh, W., & Wang, J. (2006). Efficient remote mutual authentication and key agreement. *Computers & Security, 25*(1), 72-77.

Snelick, R., Uludag, U., Mink, A., Indovina, M., & Jain, A. (2005). Large-Scale Evaluation of Multimodal Biometric Authentication using state-of-the-art Systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 27, 450-455.

Stallings, W. (2003). Cryptography and network security – principles and practices. *3rd ed. Prentice Hall.*

Steiner, J. G., Neuman B. C., & Schiller, J. I. (1988). Kerberos: An authentication service for open network systems, *Proc. Winter Usenix Conference*, 191-201.

Tsuji, T. & Shimizu, A. (2004). One-Time Password Authentication Protocol against Theft Attacks. *IEICE TRANSACTIONS on Communications, 87*(3), 523-529.

Wu, T. (1998). The Secure Remote Password Protocol. *Proceedings of the Internet Society Symposium on Network and Distributed System Security,* 97-111.

Yang, W. H., & Shieh, S. P. (1999). Password authentication schemes with smart card. *Computers and Security*, *18*(8), 727-733.

APPENDIX

The Pseudo-Codes of Implementation

The following algorithm converts an integer to its equivalent string. Since remembering strings more easy than integers, we convert integers to strings.

```
String Convert_Integer_to_String(Biginteger b)
    x = 325
    index = 0
    value = 1
    for i = 120 downto 0
        value = b / Exponentiation(x,i)
        if value = = 0
            continue
        else
            index = i
            break
    Let data[0..index] be a new char array
    k = 0
    for j = index downto 0
        value = b / Exponentiation(x,i)
        b = b – (value * Exponentiation(x,i))
        data[k] = (char) value
        k++
    return data
```

The following algorithm returns the primality of a given integer. To generate individual passwords, we modulo unique password by primes. In our study, we use the following algorithm to test randomly generated integers being prime.

```
String miller_rabin(Biginteger n)
    if n < 2
```

```
        return "composite"
if n != 2 and (n % 2) = = 0
        return "composite"
if n != 5 and (n % 5) = = 0
        return "composite"
Find integers k, q with k > 0 , q odd, so that (n – 1) = 2^k * q
Let a[0..9] be an array which includes values 2,3,5,7,11,13,17,31,61,73
for i = 0 to 9
        if Power_and_Mod(a[i],q,n) = = 1
                return "inconclusive"
        for j = 0 to k-1
                if Power_and_Mod(a[i] , Exponentiation(2, j)*q , n) = = n – 1
                        return "inconclusive"
return "composite"
```

The following algorithm calculates the value of r in equation (6).

```
Biginteger Find_M_Value(Biginteger[] primes)
M_value = 1
for i = 0 to primes.Length-1
        M_value = M_value * primes[i]
return M_value
```

The following algorithm is used to compute the value of M_i in equation (7).

```
Biginteger[] Find_Mi_Values(Biginteger[] primes)
Let Mi_values[0..primes.Length-1] be a new Biginteger array
for i = 0 to primes.Length-1
        Mi_values[i] = 1
for j = 0 to primes.Length-1
```

```
        for k = 0 to primes.Length-1
            if (j = = k)
                continue
            Mi_values[j] = Mi_values[j] * primes[k]
    return Mi_values
```

The following algorithm converts a string to its equivalent integer. This algorithm is used in our study to make mathematical operations possible for passwords.

```
Biginteger Convert_String_to_Integer(String s)
    x = 127
    Integer_value = 0
    for i = 0 to s.Length-1
        Integer_value = (x * Integer_value) + s[i]
    return Integer_value
```

The following algorithm performs the operation of x^n. Exponentiation is necessary for the algorithms Power_and_Mod( ) and miller_rabin( ).

```
Biginteger Exponentiation(Biginteger x, Biginteger n)
    if n = = 0
        return 1
    if n = = 1
        return x
    if (n % 2) = = 0
        return Exponentiation(x*x , n/2)
    else
        return Exponentiation(x*x , n/2)*x
```

Let a, n and m be positive integers. Then, the following algorithm computes a^n mod m. This algorithm is used in miller_rabin algorithm.

```
Biginteger Power_and_Mod(Biginteger number, Biginteger Power, Biginteger
mod_num)
    Let A[0..49] and mod[0..49] be new arrays
    Let M[0..14] be an array which includes values from 2^0 to 2^14
    j = 0, global_index = 0, result = 1, abc = 0
    for i = 0 to M.Length-1
        if M[i] < Power
            continue
        elseif M[i] > Power
            Power = Power – M[i-1]
            A[j] = M[i-1]
            j = j + 1
            abc = j
            i = -1
            if (Power = = 0) break
        else
            A[j] = M[i]
            break
    for index = 0 to M.Length-1
        if M[index] = = A[0]
            global_index = index
            break
    for k = 0 to global_index
        if k = = 0
            mod[k] = number % mod_num
        else
            mod[k] = Exponentiation(mod[k-1], 2) % mod_num
    for index2 = 0 to M.Length-1
        if M[index2] = = A[abc]
```

```
                result = result * mod[index2]
                result = result % mod_num
                abc = abc – 1
                if abc = = -1
                        break
        return result
```

The following algorithm checks not only the strength of the password but also its writability via standart characters which resides on keyboard.

```
Bool Control_password(string data)
        punctuation = 0, upper = 0, lower = 0, digit = 0
        for i = 0 to data.Length-1
                a = data[i]
                if (a > = 65 && a < = 90)
                        upper = upper + 1
                if (a > = 97 && a < = 122)
                        lower = lower +1
                if (a > = 48 && a < = 57)
                        digit = digit +1
                if ((a > = 33 && a < = 47) || (a > = 58 && a < = 64))
                        punctuation = punctuation + 1
                if ((a > = 91 && a < = 96) || (a > = 123 && a < = 125))
                        punctuation = punctuation + 1
        if (punctuation = = 0 || upper = = 0 || lower = = 0 || digit = = 0)
                return false
        for i = 0 to data.Length-1
                if (data[i] < 33 || data[i] > 125)
                        return false
        if (data.Length < 13)
                return false
```

```
return true
```

The following algorithm controls both the strength(length) of the unique password and the characters that make up the password. The unique password can only be formed by English letters, digits, punctuation marks and mathematical operators. Also the minimum length of the unique password should be 30 characters.

```
Int Control_X_value(string data)
    for i = 0 to data.Length-1
        if (data[i] < 32 || data[i] > 125)
            return 0
    if (data.Length < 30)
        return 1
    return 2
```

The following algorithm checks whether the number is in the array primes[] or not. If the number is in the array primes[], algorithm returns true, otherwise it returns false. In our implementation, to generate the specified number of different primes, we control the current prime whether it has been added to the array or not.

```
Bool Control_prime(Biginteger[] primes, Biginteger number)
    for i = 0 to primes.Length-1
        if (primes[i] = = number)
            return false
    return true
```

The following algorithm is a recursive function which is used to determine the greatest common divisor of two integers. In our application, randomly generated integers are firstly tested with the miller-rabin algorithm. Secondly, they are tested in pairs with this algorithm to make certain the relatively primality of the integers in pairs.

```
Biginteger Euclid_alg(Biginteger a, Biginteger b)
    if (b = = 0)
        return a
    else
        return Euclid_alg(b, a % b)
```

The following algorithm returns an array which contains the specified number of randomly generated primes.

```
Biginteger[] Find_primes_for_passwords(Biginteger key_number)
    flag2 = true, k = 0
    Let prime_numbers[0. .key_number – 1] be a Biginteger array
    while(flag2)
        k = 0
        for i = 0 to key_number – 1
            while(true)
                Randomly generate a Biginteger number and call it prime
                prime_numbers[i] = prime
                if (miller_rabin(prime_numbers[i]) = = “inconclusive”)
                    break
        for i = 0 to key_number – 2
            for j = i + 1 to key_number – 1
                b1 = Euclid_alg(prime_numbers[i], prime_numbers[j])
                if (b1 ! = 1)
                    k = 1
        if (k = = 0)
```

```
            flag2 = false
    return prime_numbers
```

The following algorithm returns an array which contains the specified number of randomly generated primes such that each prime is bigger than the corresponding integer of the array p at the same index.

```
Biginteger[] Find_primes_for_passwords(Biginteger[] p , int key_number)
    flag2 = true, k = 0
    Let num[0. .key_number – 1] be a new Biginteger array
    while(flag2)
        k = 0
        for i = 0 to key_number – 1
            while(true)
                Randomly generate a Biginteger number and call it number
                num[i] = number
                if (miller_rabin(num[i]) = = “inconclusive”&& num[i]> p[i])
                    break
        for i = 0 to key_number – 2
            for j = i + 1 to key_number – 1
                b1 = Euclid_alg(num[i], num[j])
                if (b1 ! = 1)
                    k = 1
        if (k = = 0)
            flag2 = false
    return num
```

The following algorithm returns an array which includes the results of (X mod primes[i]) for 0≤ i ≤ primes.Length -1, namely numerical equivalents of individual passwords.

```
Biginteger[]Return_xi_from_X(Biginteger X, Biginteger[] primes)
    Let result[0..primes.Length-1] be a new Biginteger array
    for i = 0 to primes.Length -1
        result[i] = X % primes[i]
    return result
```

The following algorithm computes the numerical equivalent of the unique password, namely X, when we have individual passwords and the primes.

```
Biginteger Return_X(Biginteger[] prehash, Biginteger[] Mi, Biginteger[] I_of_Mi,
Biginteger[] primes, Biginteger M)
    result = 0
    for i = 0 to prehash.Length – 1
        result = result + (prehash[i] * Mi[i] * (I_of_Mi[i] % primes[i]))
    result = result % M
    return result
```

The following algorithm tests the array primes[] if it includes the specified number of different primes.

```
Bool Control_primes_for_diversity(Biginteger[] primes, int number)
    count = primes.Length
    for i = 0 to primes.Length – 2
        for j = i + 1 to primes.Length – 1
            if (primes[i] ! = 0 && primes[i] = = primes[j])
                primes[j] = 0
```

```
                        count = count -1
      if (count > = number)
            return true
      return false
```

The following class is used for storing the values of the quotient and the remainder after division operation.

```
Class ReturnTwoValues
{
      Biginteger q  // quotient
      Biginteger r  // remainder
}
```

Let a and b be integers, then the following algorithm finds integers x and y such that x is the multiplicative inverse of a modulo b, and y is the multiplicative inverse of b modulo a.

```
ReturnTwoValues Find_Inverse(Biginteger a, Biginteger b)
      Let rt, retv, result be ReturnTwoValues objects
      if b = = 0
            rt = new ReturnTwoValues(1, 0)
            return rt
      else
            rt = Divide(a, b)
            retv = Find_Inverse(b, rt.r)
            result = new ReturnTwoValues(retv.r , retv.q – rt.q * retv.r)
            return result
```

The following algorithm returns a ReturnTwoValues object when integers x and y are given. The object includes the quotient and the remainder of the operation x/y.

```
ReturnTwoValues Divide(Biginteger x, Biginteger y)
    quotient = x / y
    remainder = x – (quotient * y)
    rt = new ReturnTwoValues(quotient, remainder)
    return rt
```

In our implementation, it is necessary for our method to obtain the multiplicative inverse of (M_i modulo p_i) for $0 \leq i \leq M_i$.Length-1. For this purpose, the following algorithm is used.

```
Biginteger[] Return_inverse_of_Mi( Biginteger[] Mi , Biginteger[] mi)
    Let result[0..Mi.Length-1] be a new Biginteger array
    Let rt1 be a ReturnTwoValues object
    for i = 0 to Mi.Length-1
        rt1 = Find_Inverse(Mi[i] , mi[i])
        result[i] = rt1.r
    return result
```

The following algorithm generates individual passwords and primes in accordance with the Forward Direction method.

```
Forward_Direction_Method(string password , int num)
    Xvalue = Convert_String_to_Integer(password)
    Let primes[0..num-1] be a new Biginteger array
    flag2 = true, index = 0
    while(flag2)
        newprime = Find_primes_for_passwords(5)
```

```
        x1 = Convert_Integer_to_String(Xvalue % newprime[0])
        if (Control_password(x1) && Control_prime(primes , newprime[0]))
            primes[index] = newprime[0]
            index = index + 1
            if (index = = num && Control_primes_for_diversity(primes,num))
                flag2 = false
    for j = 0 to number – 1
        passwords[j] = Convert_Integer_to_String( Xvalue % primes[j])
    Print passwords and primes to the screen
```

The following algorithm generates primes for the given passwords and computes the unique password in accordance with the Backward Direction method.

```
Backward_Direction_Method(string[] passwords)
    num = passwords.Length
    Let P_o_s[0..(num*2)-1] be a new Biginteger array
    for i = 0 to num-1
        P_o_s[i] = Convert_String_to_Integer(passwords[i])
    for j = num to (num*2)-1
        Randomly generate a Biginteger value and call it number
        P_o_s[j] = number
    flag1 = true
    Let primes[0..(num*2)-1] be a new Biginteger array
    Let Mi_values[0..(num*2)-1] be a new Biginteger array
    Let I_o_Mi[0..(num*2)-1] be a new Biginteger array
    Let xi[0..(num*2)-1] be a new Biginteger array
    X_value = 0
    M_value = 0
    while(flag1)
        primes = Find_primes_for_passwords(P_o_s , num*2)
```

```
        Mi_values = Find_Mi_Values(primes)
        M_value = Find_M_Value(primes)
        I_o_Mi = Return_inverse_of_Mi(Mi_values, primes)
        X_value = Return_X(P_o_s, Mi_values, I_o_Mi, primes, M_value)
        xi = Return_xi_from_X(X_value , primes)
        if (P_o_s[0] = = xi[0])
            flag1 = false
    X = Convert_Integer_to_String(X_value)
    Print X, passwords and primes to the screen
```